Eric Tang 705579803
Yolanda Chen 005323226

CS 152a Final Project Report

1. **Introduction and requirement (10%). Summarize background information about the lab and the detailed design requirements. It's very important to make sure you are designing the right thing before starting.**

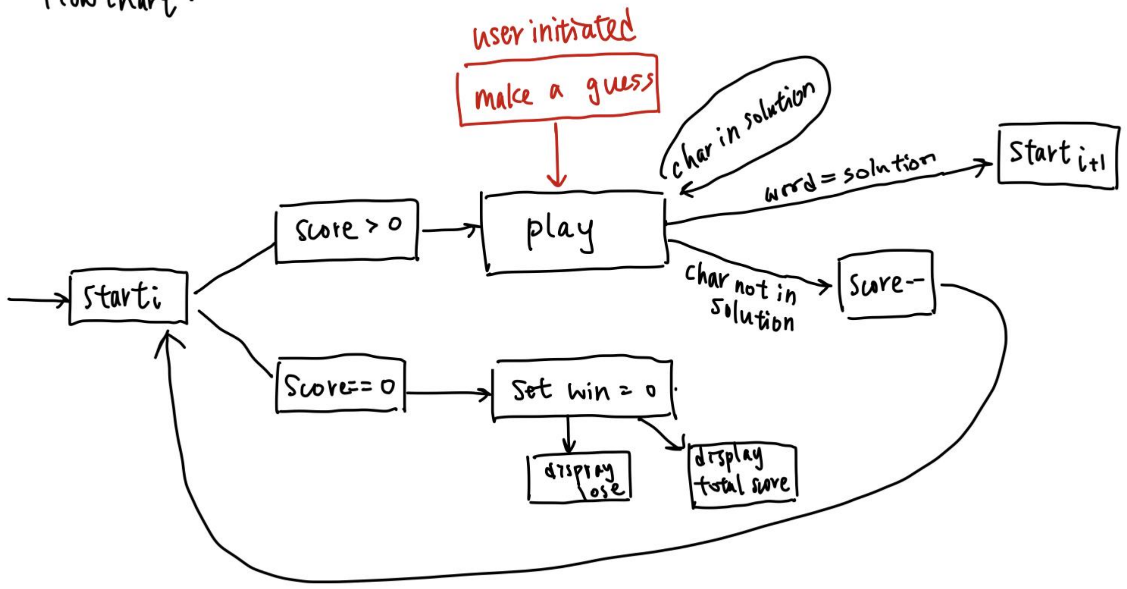
For this lab, we use the Xilinx ISE software to create the hangman game on the Nexys™3 Spartan-6 FPGA Board with Pmod SSD. The user is able to guess letters in a four-letter-word by toggling switches to form their five-bit binary representations (e.g. the letter "a" is represented by 00000, "b" is represented by 00001). The user starts off with 10 points on each level. The seven-segment display starts off with four dashes (unguessed characters will be displayed as dashes). A correct guess will reveal the corresponding letter(s). A wrong guess will decrement the points of the current level by 1. If it reaches 0, the player loses the game. Their final score will be displayed on the Pmod SSD along with the "LOSe" message on the seven-segment display. This behavior is observed in the video titled "lose.MOV". If the player clears the current level (i.e. guess all the letters in the word correctly before the score reaches 0), the remaining score will be added to the final score and the player moves on to the next level. If the player completes all the levels without losing, the Pmod SSD will display the final score and the seven-segment display will display "YeAH". This behavior is observed in the video titled "win.MOV".

2. **Design description (15%). Document the design aspects including the basic description of the design, modular architecture, interactions among the modules, and interface of each major module. You should include schematics for the system architecture. You can also include figures for state machines and Verilog code when needed.**

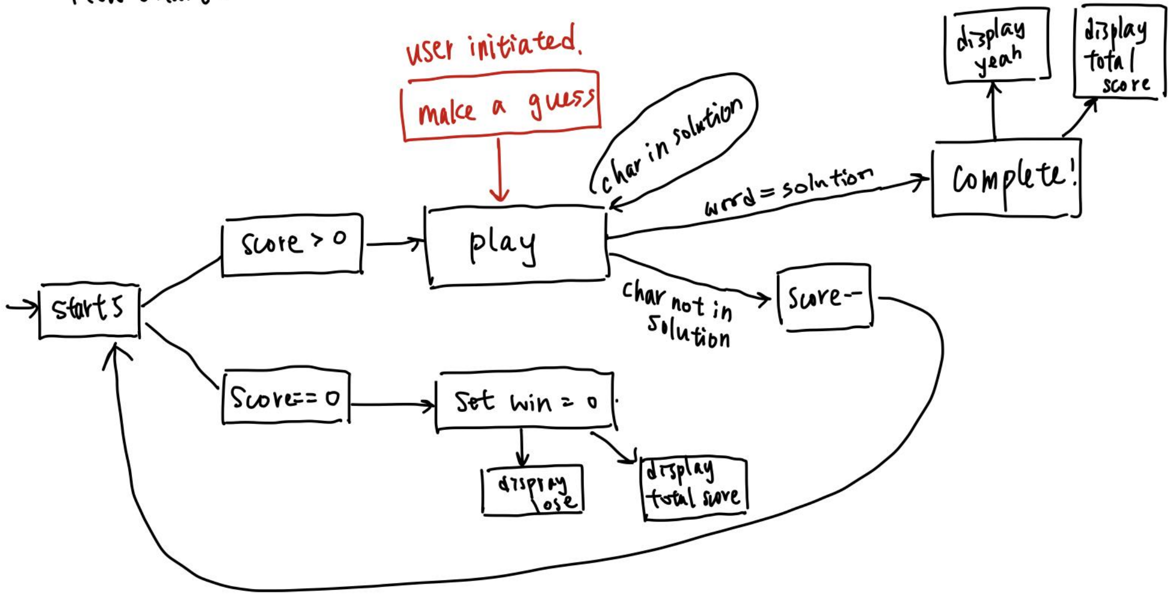
The following is the schematics of our hangman game:

Eric Tang 705579803
Yolanda Chen 005323226

Level 1-4
Flow chart:



Level 5
Flow Chart:



Seven Segment Display:
There are four seven segment displays on our FPGA board. We use them to display the words for the hangman game. Since there are only four displays, our hangman questions consist of only four-letter words.

Eric Tang 705579803
Yolanda Chen 005323226

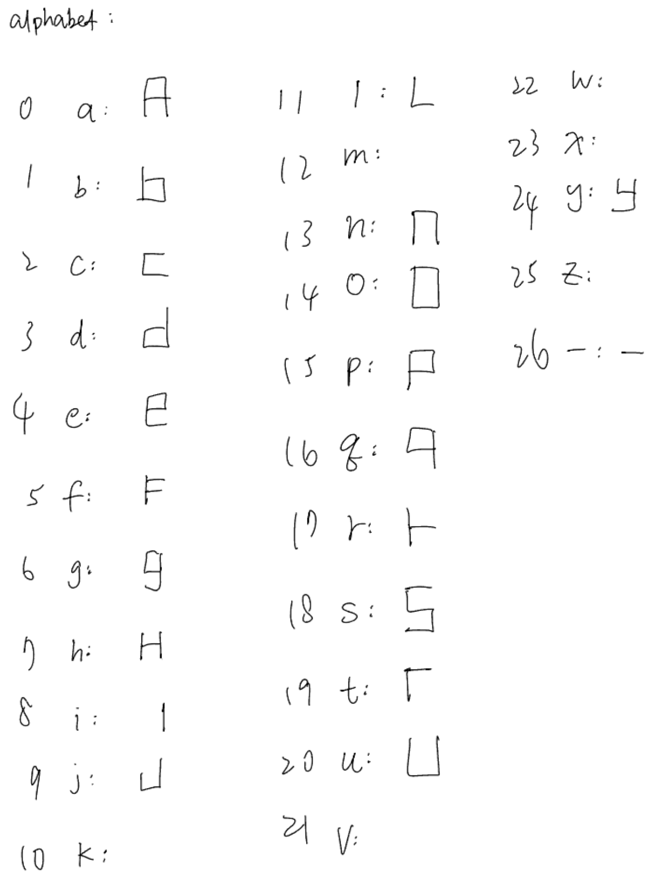
From left to right, we named each seven segment display as **seg3**, **seg2**, **seg1**, and **seg0**. If we want to display the word "UCLA", then **seg3** = "U", **seg2** = "C", **seg1** = "L", and **seg0** = "A". Note that **seg3**, **seg2**, **seg1**, and **seg0** stores the numerical value that represents each letter and not the character itself. In the following discussions, we will explain more about the encoded numerical value for each letter in the alphabet.

At the start of the game, the four displays will show "- - - -" (Dashes represent unguessed characters). The player is able to send their guesses to the program using the switches and push button (switches and push button designs will be discussed later on in this section) on the FPGA. If the player guesses correctly, the corresponding display will reveal the correctly guessed letter(s). For example, if the answer is "UCLA" and the player guesses "A", then the display will show "- - - A". If the player guesses incorrectly, the display should stay the same.

Since the seven segment display is meant for displaying numbers, we faced difficulties displaying every letter in the alphabet. Our display is limited to certain characters in the English alphabet. For example, we aren't able to display uppercase or lowercase K, M, W, V, Z and X.

The below is our design for letters we can display with the seven segment displays and their corresponding numerical value that will be stored in **seg3**, **seg2**, **seg1**, and **seg0** when used:

Eric Tang 705579803
Yolanda Chen 005323226



To display the word correctly, we set each letter separately in every cycle of **segment_clk** (a very fast clock) and display them in order over multiple cycles. Since **segment_clk** is very fast, the player cannot tell that each letter is actually displayed individually. We accomplished this by creating a 2-bit variable **digit_dis** that increments by one in every cycle of **segment_clk.** Since it is a 2-bit value, **digit_dis** only has values of 00, 01, 10, and 11. Using **digit_dis**, our code will be able to turn on the corresponding seven-segment display in each cycle.

Switches and Guessing Letters:
The player needs to toggle 5 switches in order to guess a letter every turn. We decode all the available letters to numbers. The player needs to input the corresponding binary value using the switches to select the letter they want to send.

Pmod SSD and JA JB Inputs:

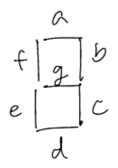
Eric Tang 705579803
Yolanda Chen 005323226

Besides displaying the words in our hangman game, we also need a score display. Therefore, we used the Pmod SSD to display the player’s current score during the game.

To use the Pmod SSD, we included 2 new inputs, **JA** and **JB**. **JA** and **JB** are both 8-bit inputs. Each of the four lower bits of **JA** corresponds to a segment on the number display. Each of the three lower bits of **JB** also corresponds to a segment on the number display. The fourth lower bit of **JB** controls which of the two number displays is turned on in each cycle. Similar to the four seven segment displays on the FPGA, we can only display one number on the Pmod SSD in each cycle. To create the illusion that the two number displays are displayed at the same time, we alternate **JB**[3] between 0 and 1 every **segment_clk** cycle.

The below are the values of the first four bits of **JA** and **JB** for each digit:

	JA abcd	JB efgx
0:	0000	0010
1:	1001	1110
2:	0010	0100
3:	0000	1100
4:	1001	1000
5:	0100	1000
6:	0100	0000
7:	0001	1110
8:	0000	0000
9:	0001	1000



Score Calculation:
As mentioned above, the two digit displays on the Pmod SSD are used to display the current score of the player. We created 2 variables,

Eric Tang 705579803
Yolanda Chen 005323226

score0 (for the left display) and **score1** (for the right display), to store the number each digit display should display at any moment.

At the start of each level, the player has 10 points. Each time the player guesses incorrectly, the current score is decremented by 1. The remaining score after completing each level will be added to the total score. The total score will be displayed after the player completes the game (by either winning or losing the game).

Below is a pseudocode of how **score0** and **score1** are calculated:

```
// game ends: player loses or completes the game
if (player loses or wins the entire game) begin
    score1 = total_score % 10;
    score0 = total_score / 10;
end
// game continues: player is still playing a level
else begin
    score1 = cur_score % 10;
    score0 = cur_score / 10;
end
```

Debouncing and Metastability:
To deal with debouncing and metastability, we created a separate 2-bit variable for each of the switches and buttons that is responsible for checking if the user actually presses the button or turns on the switch. For example, for the reset button, we created a variable called **setButton**. **setButton** works similar to a 2-bit predictor. It sets **send** to True if it detects that the button **btnR** is high twice in a row. By doing so, we eliminate the possibility of changing states because of unstable signals.

Our pseudocode for debouncing and metastability is shown below:

```
input btnR
reg send;
reg [1:0] setButton;

// for each sampling point
always @(posedge segment_clk) begin
    shift setButton right by 1
    if (btnR is high)
        Set msb of setButton to 1
    if (setButton == 2'b11)
        Set send to True
    else
```

Eric Tang 705579803
Yolanda Chen 005323226

```
                Set send to False
    end
```

User Constraint File (nexys3.ucf):
For this lab, we had to use several components of the FPGA. Here is the list of components we used and their purposes:

clk	The 100MHz base clock for generating the fast **segment_clk.**
bntR	The right button on the FPGA that is used for sending the player's guess.
sw<0>-<4>	Represents each bit for a 5-bit binary value. Used for selecting letters.
an<0>	The seven-segment number display that displays the value of **seg0.**
an<1>	The seven-segment number display that displays the value of **seg1.**
an<2>	The seven-segment number display that displays the value of **seg2.**
an<3>	The seven-segment number display that displays the value of **seg3.**
JA JB	Connects FPGA to Pmod SSD.

3. **Conclusion (5%). Summary of the design. Difficulties you encountered, and how you dealt with them. General suggestions for improving the lab, if any.**

We've encountered several difficulties in the final project. They can be broken down to two categories: the confusion of FPGA/display setup and coding errors.

First, we were trying to display the words using the Uart display. However, after reading through the code in lab 1 and tweaking it, we realized that the logic was quite complicated and it wouldn't be feasible to achieve what we wanted with the display. Afterwards, we decided to display words using a seven-segment display and display the score using Pmod SSD. This approach was more straightforward although it limits our word length to be 4. We also had some trouble figuring out the digit display of the Pmod SSD because it involved setting two arrays.

Eric Tang 705579803
Yolanda Chen 005323226

After the setup, we encountered some other errors. We observed that whenever the player guessed wrong, the current level score reached 0 immediately. We found out that it was because the score decrementing logic was run at a very high frequency when the **btnR** is 1, so a wrong input can trigger it multiple times. We fixed this by creating a **always** block that's only triggered on the positive edge of **btnR** and put the decrementing logic in there. We also incorporate debouncing with **btnR** to prevent noise from setting the button. Later on, we encountered some other logic errors such as not displaying dashes when moving on to the next level and not displaying the entire word when the user guessed all letters correctly. But overall, they're minor mistakes and didn't take us long to fix.